Répartition dynamique de données régulières pour des machines MIMD homogènes à mémoire distribuée[†]

Mihaela Juganaru, Ibrahima Sakho

Ecole Nationale Supérieure des Mines de Saint Etienne, Centre SIMADE
158, cours Fauriel, 42023 Saint Etienne Cedex 2
Tél. : (33) 77 42 66 03
Fax : (33) 77 42 66 66
E-mail : juganaru, sakho@emse.fr

RÉSUMÉ. *Dans ce papier nous présentons une stratégie simple pour la répartition dynamique de charges de données régulières sur des machines homogènes MIMD à mémoire distribuée et à réseau d'interconnexion statique. C'est à dire des données dont chaque unité peut être traitée indépendamment et exige le même temps de traitement sur chaque processeur. La stratégie est basée sur une connaissance globale de la distribution de la charge de données acquise grâce à un calcul de préfixe généralisé sur un arbre recouvrant du graphe du réseau d'interconnexion. Elle est donc utilisable pour toute topologie. Nous montrons que la politique d'échanges de données induite par cette stratégie est de complexité en temps* $O(\min_{P^*} \|P^*\| K(P^*))$ *où* P^* *est un chemin de poids maximum de l'arbre recouvrant considéré,* $\|P^*\|$ *sa longueur,* $K(P^*)$ *le poids maximum de ses arcs.*

ABSTRACT. *This paper reports a simple strategy for dynamically load balancing of regular data on homogeneous Distributed Memory MIMD computers with static interconnexion network. That is data in which each unit can be processed independently and requires the same time on each processor. This strategy is based upon a global knowledge of the initial distribution of the data' load obtained by a generalized prefix computation on a spanning tree of the graph of the interconnexion network of the processors; this strategy is then usable for any topology. We prove also that the data exchanges induced by this strategy lead to the balance of the data' load in time* $O(\min_{P^*} \|P^*\| K(P^*))$*, where* P^* *is a maximum weight path of the spanning tree,* $\|P^*\|$ *its length and* $K(P^*)$ *the maximum weight of its arcs.*

MOTS CLÉS : *machine MIMD à mémoire distribuée, allocation dynamique, algorithme distribué, calcul de préfixe*
KEYWORDS :*Distributed Memory MIMD computers, Dynamic load balancing, distributed algorithm, prefix computing*

[†] Ce travail est soutenu par la Région Rhône–Alpes dans le cadre du projet "Nouvelles architectures parallèles et développement d'applications"

1 Introduction

L'allocation de la charge de travail des applications parallèles aux processeurs est un élément essentiel pour une utilisation efficace des machines parallèles. En effet elle influe non seulement sur le temps total d'exécution des applications, mais aussi dans le cadre d'une multiprogrammation sur le taux d'occupation des processeurs. L'allocation de charges, quel qu'en soit l'objectif, peut se faire de manière statique ou de manière dynamique. Lors d'une allocation statique les charges sont allouées une fois pour toutes avant l'exécution de l'application, tandis que lors d'une allocation dynamique, les charges peuvent être réallouées pendant l'exécution pour répondre à certains critères d'efficacité.

Une application parallèle est un ensemble de processus communicants modélisé généralement par un graphe dont les nœuds correspondent aux processus et les arêtes aux communications inter–processus. Quand elle est bien connue, c'est à dire le temps d'exécution de chaque processus et le temps de communication de tout couple de processus communicants sont connus, l'allocation statique est plus appropriée. Mais, en général, ces paramètres ne sont pas assez connus; le temps d'exécution d'un processus peut, par exemple, varier suite à une génération spontanée de nouveaux processus modifiant ainsi le schéma et le temps de communication de l'application. L'allocation dynamique qui permet de prendre en compte de tels comportements imprédictibles est alors plus appropriée. Une taxonomie et une classification des méthodes d'allocation dynamique peuvent être trouvées dans [CAS88A] et dans des papiers plus récents tels que [SHI92], [BER91].

Dans ce papier nous nous intéressons à l'allocation dynamique de charges constituées de données régulières sur des machines MIMD homogènes à mémoire distribuée et à réseau d'interconnexion statique. Ceci signifie que chaque unité de données engendre sur n'importe quel processeur le même temps de traitement; en plus, ces traitements sont indépendants. Cette hypothèse qui semble être assez restrictive couvre en fait un vaste ensemble d'applications très exigeantes en puissance de calcul. C'est, par exemple, le cas pour les simulations moléculaires dans la physique des particules [BRU90], [BOI91], pour le traitement d'images [MIG92], [GER93], pour la mécanique des fluides [HEI94], etc. Pour de telles applications le processus d'allocation de charges consiste plutôt en celui de la régulation de charges, i.e. le maintien du même nombre d'unités de données sur chaque processeur.

Quand on fait de l'allocation dynamique de charges sur une machine MIMD à mémoire distribuée, il y a trois questions auxquelles on doit répondre : quand initier le processus, par qui et selon quelle stratégie. Les deux premières questions connues aussi sous le nom de mécanismes (politiques) d'allocation sont amplement traitées dans la littérature [ZAT85], [CAS88B], [SHI92]. Dans ce papier nous traiterons uniquement des stratégies.

Les stratégies utilisent toutes l'une des formes de connaissance de la distribution initiale de la charge. Aussi dans la littérature distingue-t-on les stratégies basées sur une connaissance locale et celles basées sur un connaissance globale de cette distribution. On parlera alors de stratégies avec information locale et de stratégies avec information globale.

Dans les stratégies avec information locale, aussi connues sous le nom de stratégies "diffusives", de manière itérative, chaque processeur diffuse son état (i.e. le montant de sa charge) à ses voisins, puis calcule le montant de la charge à échanger avec eux suivi d'un échange effectif des charges. En général, pour chaque processeur, cette charge est calculée selon un certain seuil variable déterminé en fonction de sa propre charge et celle de ses voisins. Cette approche, proposée initialement par Cybenko

[CYB89] et Boillat [BOI90] et plus récemment reprise par Heirich [HEI94], assimile donc le processus de régulation de charges à l'évolution d'un système dynamique. La convergence du système vers un état d'équilibre est alors régie par les propriétés spectrales de l'opérateur d'évolution, généralement choisi linéaire.

Il est fréquent lors de l'évolution d'un tel système d'avoir à manipuler des quantités non-entières de charges à échanger privant ainsi les résultats de toute réalité. Subramanian et Scherson dans [SUB94] proposent quelques modifications de l'algorithme initial de Cybenko pour pallier cette insuffisance.

Tandis que Cybenko et Boillat établissent la complexité de leur algorithme à $O(N^2)$ pas d'itération, où N est le nombre de processeurs, Subramanian et Scherson [SUB94] donnent une complexité plus fine en temps de travail. Pour un réseau de N nœuds, de conductance électrique Γ, de conductance du fluide Φ, ils prouvent que lorsque le déviation de la distribution de charge initiale vaut σ, le temps $Time$ nécessaire pour équilibrer la charge des processeurs vérifie $\Omega(log\sigma/\Gamma) \leq Time \leq O(N\sigma/\Gamma)$ et $\Omega(log\sigma/\Phi) \leq Time \leq O(\sigma/\Phi^2)$.

Bien que naturellement distribuées, indépendantes de la topologie du réseau d'interconnexion et permettant une exploitation de tous les liens de communication inter–processeurs, les stratégies diffusives exigent beaucoup de communications pour le contrôle du processus d'allocation et pour l'échange de charges proprement dit.

Les stratégies avec information globale nécessitent comme annoncé précédemment une connaissance globale de la distribution initiale de la charge à répartir. Elles consistent d'abord en l'évaluation des quantités de données que les processeurs doivent échanger, puis en l'échange effectif des charges correspondantes. Elles utilisent pour cela une technique de type calcul de préfixe développée par Blelloch [BLE89].

Jusque là utilisées pour des topologies telles que le réseau linéaire [MIG92], [GER93] ou l'hypercube [JAJ92], les stratégies de cette classe exploitent la forte régularité du réseau d'interconnexion des processeurs.

La stratégie que nous proposons dans ce papier est aussi avec information globale, mais indépendante de la topologie d'interconnexion des processeurs. Elle comprend trois phases. La première phase consiste à extraire, par un parcours en largeur d'abord à partir du centre, un arbre recouvrant du graphe du réseau d'interconnexion; l'information globale sur la distribution de la charge et l'évaluation des charges à échanger sont obtenues après, à l'aide d'un calcul de préfixe généralisé. La troisième phase décrit le protocole d'échanges des charges. Nous montrons que la politique d'échanges de données ainsi obtenue, qui indique à tout couple de nœuds la quantité de données qu'ils doivent échanger, induit un équilibrage de charges proprement dit de complexité $O(\min_{P^*} \|P^*\| K(P^*))$ où P^* est un chemin de poids maximum de l'arbre recouvrant, $\|P^*\|$ sa longueur, $K(P^*)$ le poids maximum de ses arcs.

Cet algorithme n'induit pas de manipulation de valeurs non-entières de charges à échanger. Donc les problèmes d'instabilité inhérents à bien d'allocateurs dynamiques de charges n'apparaissent pas. La donnée d'une quantité de charges à échanger sur une connexion oriente de façon unique celle-ci, évitant ainsi le phénomène de "ping pong" des données inhérent également aux méthodes diffusives et de nature à dégrader les performances des allocateurs. Des protocoles d'échanges efficaces des données peuvent alors être envisagés.

Le reste du papier est organisé de la manière suivante. La section 2 donne une description complète de l'algorithme. Dans la section 3 la convergence de la phase d'échanges des données et sa complexité en temps sont analysées. Dans la section 4 nous discutons des résultats des simulations faites sur une machine MIMD à base de transputers pour des grilles toriques.

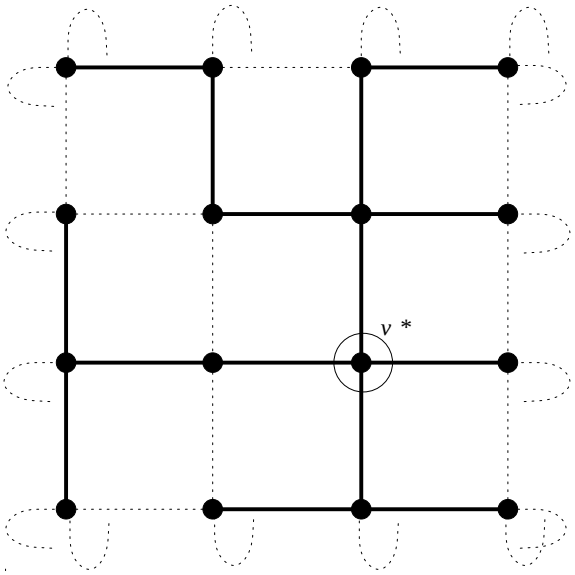

Figure 1. *Un arbre recouvrant selon un parcours en largeur d'abord pour la grille torique* 4×4*,* v^* *désigne la racine et les arêtes sont représentées en gras.*

2 Description de l'algorithme

L'algorithme que nous proposons est distribué et consiste en trois phases. Soit $G = (V, E)$ le graphe du réseau d'interconnexion des processeurs où V est l'ensemble des nœuds qui représentent les processeurs et E est l'ensemble des arcs qui représentent les liens d'interconnexion des processeurs. La première phase de l'algorithme calcule, à partir du centre v^* de G et ce une fois pour toutes, un arbre recouvrant par un parcours en largeur d'abord. Nous ne développerons pas ici un tel calcul qui est très amplement décrit dans la littérature (voir par exemple [KOR84], [GAL83] ou [LAV95]). Soit alors $T = (V, E')$ l'arbre recouvrant, enraciné en v^*. La figure 1 donne un tel arbre pour la grille torique 4×4; n'importe quel nœud peut être choisi comme centre car les nœuds sont indifférenciés.

Pour décrire la deuxième phase nous adopterons les définitions et les notations suivantes :

- $v^{(0)}$ pour le père du nœud v;
- T_v pour le sous-arbre de T enraciné en v; si v est une feuille de T, T_v est alors réduit au nœud v;
- $taille(T_v)$ pour le nombre de nœuds du sous-arbre T_v;
- $charge(x)$ est la charge de x où x est soit un nœud, soit un sous-arbre de T; dans ce dernier cas $charge(x)$ est égale à la somme des charges de tous ses nœuds.

Comme annoncé dans l'introduction lorsque la charge est constituée de données régulières et que les processeurs de la machine de traitement sont homogènes, le but

du processus de régulation de charges est de distribuer la charge totale du réseau, $charge(T_{v^*})$, tel que chaque processeur ait approximativement la même quantité de charges. Plus précisément tout processeur devrait avoir $\lfloor charge(T_{v^*})/|V| \rfloor$ ou $\lceil charge(T_{v^*})/|V| \rceil$ unités de charges à la fin de la distribution.

Pour cela un calcul de préfixe [BLE89] de type généralisé est initialisé en chaque nœud. A la suite d'un tel calcul, chaque nœud v connaît $charge(T_v)$ et $taille(T_v)$, $charge(T_u)$ et $taille(T_u)$ pour tout nœud fils u. En particulier, la racine v^* est à même d'évaluer la charge moyenne $\overline{charge}$ de tout le réseau et de la diffuser vers tous les nœuds. L'étape suivante consiste, pour tout nœud v à déterminer, par rapport à la charge moyenne, l'état de la charge de T_v, ainsi que T_u pour chacun de ses fils u. Un des trois cas suivants peut apparaître :

1. $charge(T_x) > taille(T_x) * \overline{charge}$, alors T_x est surchargé. Il doit se délester de son excédent de charges en envoyant celui-ci à l'extérieur du sous-arbre, par exemple à $x^{(0)}$ à travers la racine x.

2. $charge(T_x) < taille(T_x) * \overline{charge}$, alors T_x est sous–chargé. Il doit combler son déficit grâce à un sous-arbre excédentaire par exemple par le biais de $x^{(0)}$.

3. $charge(T_x) = taille(T_x) * \overline{charge}$, alors T_x est globalement équilibré. La régulation de la charge de ses nœuds est une opération interne à ce sous-arbre. Cette opération est récursivement réalisée selon l'état de la charge de chaque sous-arbre enraciné dans les nœuds fils comme décrit dans les deux premiers cas.

En fait les deux premiers cas orientent chaque arête de T_{v^*} et lui attachent la quantité de données que le nœud origine devra envoyer au nœud extrémité. Chaque nœud dispose ainsi d'un ensemble de nœuds qui émettent vers lui et d'un ensemble de nœuds vers lesquels il émet. La figure 2 donne une illustration de ces échanges pour l'arbre recouvrant de la figure 1 extrait de la grille torique 4×4.

Une question importante est alors comment réaliser correctement ces échanges de charges sous la contrainte naturelle qu'un nœud v ne peut envoyer plus que la charge qu'il détient. Plusieurs protocoles peuvent être utilisés. Celui que nous proposons est basé sur le principe suivant : la charge qu'un nœud v enverra à un voisin u devra délester T_v de sa surcharge, si $u = v^{(0)}$, ou combler le déficit de T_u, si u est un fils. Autrement dit, l'échange de données entre deux voisins se fait en une unique communication.

Ceci se traduit par le fait que tout nœud émetteur qui détient une charge quelconque doit d'abord s'assurer que celle-ci couvre au moins les besoins d'un de ses voisins récepteurs. Dans la négative il se met en attente de données de la part de ses voisins émetteurs avant d'essayer à nouveau de couvrir les besoins d'un voisin récepteur.

Pour décrire plus formellement l'activité d'un nœud v pendant le processus de régulation de charges, soit S, (respectivement R) l'ensemble de ses voisins qui émettent vers lui (respectivement vers lesquels il émet) et soit $echange(x, y)$ le montant de la charge que x doit envoyer à y. L'algorithme s'écrit comme suit :

/* 1-ère phase */
Calculer l'arbre recouvrant T_{v^*} enraciné au centre v^* du graphe d'interconnexion.

/* 2-ème phase : calcul de la charge moyenne $\overline{charge}$ */
/* à l'aide du calcul de préfixe généralisé sur T_{v^*} */
(1) $taille(T_v) \leftarrow 1$;
$charge(T_v) \leftarrow charge(v)$;

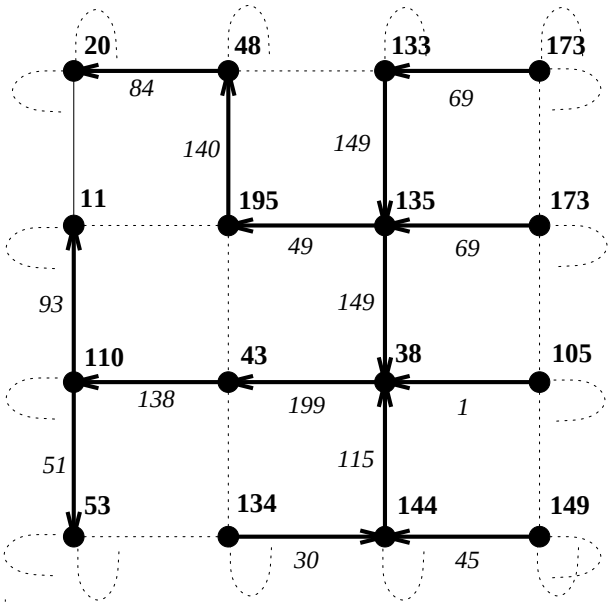

Figure 2. *Schéma de distribution de charges dans une grille torique* 4×4*. La distribution initiale de la charge est représentée par les nombres en gras. Les nombres en italique sur les arcs indiquent la quantité de données que le nœud origine doit envoyer au nœud extrémité.*

(2) pour tout u fils de v
  Recevoir $taille(T_u)$ et $charge(T_u)$ de u;
  $taille(T_v) \leftarrow taille(T_v) + taille(T_u)$;
  $charge(T_v) \leftarrow charge(T_v) + charge(T_u)$;
(3) Envoyer $taille(T_v)$ et $charge(T_v)$ à $v^{(0)}$;
/* 2-ème phase : diffusion $\overline{charge}$ */
(4) Si $v \neq v^*$ Recevoir $\overline{charge}$ de $v^{(0)}$;
(5) pour tout u fils de v
  Envoyer $\overline{charge}$ à u;
/* 2-ème phase : calculer les échanges de charges */
(6) $R, S \leftarrow \emptyset$;
(7) Si $charge(T_v) > taille(T_v) * \overline{charge}$ alors
  $R \leftarrow R \cup \{v^{(0)}\}$;
  $echange(v, v^{(0)}) \leftarrow charge(T_v) - taille(T_v) * \overline{charge}$;
  Si $charge(T_v) < taille(T_v) * \overline{charge}$ alors
  $S \leftarrow S \cup \{v^{(0)}\}$;
  $echange(v^{(0)}, v) \leftarrow taille(T_v) * \overline{charge} - charge(T_v)$;
(8) pour tout u fils de v
  Si $charge(T_u) > taille(T_u) * \overline{charge}$ alors
    $S \leftarrow S \cup \{u\}$;

$echange(u,v) \leftarrow charge(T_u) - taille(T_u) * \overline{charge}$;
Si $charge(T_u) < taille(T_u) * \overline{charge}$ alors
$R \leftarrow R \cup \{u\}$;
$echange(v,u) \leftarrow taille(T_u) * \overline{charge} - charge(T_u)$;
/* 3-ème phase : échanges de charges entre voisins */
(9) Tant que $R, S \neq \emptyset$ faire
Tant que $R \neq \emptyset$ et $\exists x \in R : charge(v) \geq echange(v,x)$
Envoyer la quantité $echange(v,x)$ de charges à x;
$charge(v) \leftarrow charge(v) - echange(v,x)$;
$R \leftarrow R - \{x\}$;
Si $S \neq \emptyset$ alors
Recevoir la quantité $echange(x,v)$ de charges de x;
$charge(v) \leftarrow charge(v) + echange(x,v)$;
$S \leftarrow S - \{x\}$;

3 Analyse de l'algorithme

La première phase est dominée par le calcul du centre v^* du graphe et par le calcul d'un arbre recouvrant minimal de racine v^*. On peut, comme indiqué précédemment, utiliser pour ces calculs un algorithme distribué comme celui décrit dans [LAV95] et dont la complexité est $O(n^2 \log n)$ en temps et $O(n^2 \log n)$ en nombre de messages de taille $O(\log n)$ bits. Il faut toutefois noter que quand même bien l'allocateur peut être sollicité plusieurs fois lors d'une même application, ces calculs ne sont effectués qu'une seule fois.

La deuxième phase est plutôt dominée par la collecte de l'information sur la distribution initiale de la charge du réseau. En raison de son caractère distribué (vague d'informations remontant depuis les feuilles jusqu'à la racine de l'arbre recouvrant et inversement), son coût est $O(h)$ en temps et $O(n)$ en nombre de messages qui sont de taille constante, h étant la profondeur de l'arbre.

La troisième phase exige au plus $n-1$ messages qui sont de taille très variable. L'analyse de cette phase, la plus coûteuse, concerne l'étude de sa correction ainsi que l'évaluation de sa complexité en temps.

Considérons le digraphe $\overrightarrow{T_{v^*}}$ construit à partir de T_{v^*} en orientant les arêtes selon le sens des échanges de données entre nœuds voisins; pour tout triplet (v, R, S) et pour tout nœud x l'arête définie par v et x est orientée de v vers x ou de x vers v, selon que, respectivement, $x \in R$ ou $x \in S$. Pour tout sommet v tel que T_v est excédentaire, on a, d'après les points (7) et (8) de l'algorithme :

$$R = \{v^{(0)}\} \cup \{u : T_u \text{ est déficitaire}\}$$
$$S = \{u : T_u \text{ est excédentaire}\}.$$

Considérons alors la grandeur

$$\sum_{u \in S} echange(u,v) - \sum_{u \in R} echange(v,u).$$

Par définition de R on a :

$$\sum_{u \in R} echange(v,u) \quad = \quad (charge(T_v) - taille(T_v) * \overline{charge})$$

$$+ \sum_{u:T_u \text{ déficitaire}} (taille(T_u) * \overline{charge} - charge(T_u))$$

avec $charge(T_v) = charge(v) + \sum_{u:T_u \text{ déficitaire}} charge(u) + \sum_{u:T_u \text{ excédentaire}} charge(u)$,

et

$$\sum_{u \in S} echange(u,v) \quad = \quad \sum_{u:T_u \text{ excédentaire}} (charge(T_u) - taille(T_u) * \overline{charge})$$

Tous calculs faits on obtient :

$$\sum_{u \in S} echange(u,v) - \sum_{u \in R} echange(v,u) \quad = \quad \overline{charge} * (taille(T_v) - \sum_u taille(T_u)) - charge(v).$$

Mais comme $taille(T_v) = 1 + \sum_u taille(T_u)$ il suit que :

$$charge(v) + \sum_{x \in S} echange(u,v) - \sum_{u \in R} echange(v,u) = \overline{charge}.$$

C'est à dire qu'après que v ait réalisé ses échanges avec ses voisins (père et fils) sa charge courante égale la charge moyenne. Par le même raisonnement on établit un résultat identique pour tout sommet v tel que T_v est déficitaire.

Evaluons maintenant la complexité en temps de l'algorithme. Dans ce but nous considérons le pire des cas de l'exécution du pas (9), c'est à dire un nœud reçoit d'abord toute la charge qu'il attend de ses voisins $u \in S$ avant d'envoyer celles qu'attendent de lui ses voisins $u \in R$, où R et S sont à leurs valeurs initiales calculées au pas (8). Nous faisons l'hypothèse que le temps de transfert d'une unité de données entre deux nœuds adjacents égale une unité de temps. Ainsi si nous désignons par $date(v)$ la date à laquelle v a reçu de tous ses voisins $u \in S$ la charge attendue, on a :

$$date(v) = \begin{cases} 0 & \text{si } S = \emptyset \\ \max\limits_{u \in S}(date(u) + echange(u,v)) & \text{sinon.} \end{cases}$$

En considérant $echange(x,y)$ comme le poids de l'arc (x,y), il suit que la date t^* à laquelle tous les sommets ont leur charge équilibrée est au plus égale au poids d'un chemin de poids maximum du digraphe $\overrightarrow{T_{v^*}}$, d'origine u telle que $S = \emptyset$ et d'extrémité v telle que $R = \emptyset$. La figure 3 donne une illustration d'un tel chemin pour le digraphe de la figure 2.

Soit alors P^* un tel chemin, $\|P^*\|$ sa longueur, et $K(P^*) = \max\limits_{(x,y) \in P^*} echange(x,y)$.

On a :

$$t^* \leq \sum_{(x,y) \in P^*} echange(x,y) \leq \sum_{(x,y) \in P^*} K(P^*) = \|P^*\| * K(P^*)$$

d'où :

$$t^* = O(\min_{P^*}(\|P^*\| * K(P^*))).$$

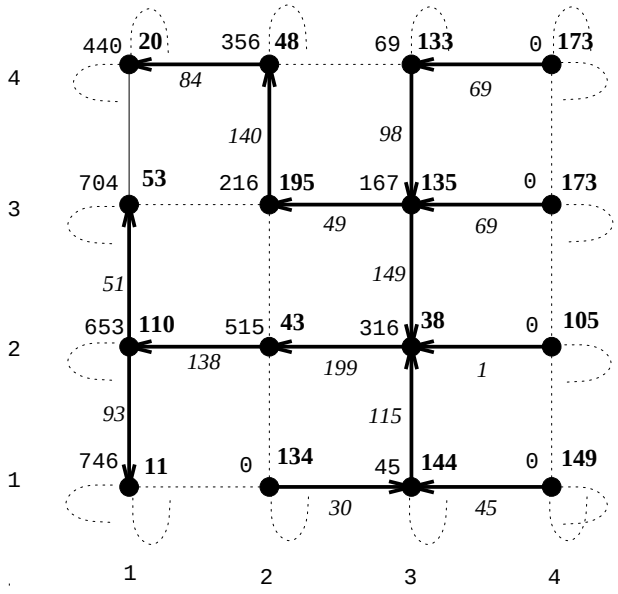

Figure 3. *Schéma d'exécution des échanges de charges. Les nombres en courrier correspondent aux dates au plus tard auxquelles les nœuds auront reçu la charge qu'ils attendent de leurs prédécesseurs dans* $\overrightarrow{T_v*}$*. Le chemin de poids maximum est donné par la suite de nœuds ((4,4) (3,4) (3,3) (3,2) (2,2) (1,2) (1,1)) de poids 746.*

4 Implantation

Dans cette section nous présentons les résultats de simulation de l'algorithme pour des grilles toriques configurées sur une VOLVOX-TS306, machine MIMD homogène reconfigurable, à mémoire distribuée, disposant de 24 processeurs de travail de type transputer T805. A cause du nombre des processeurs et de celui limité à 4 des liens physiques de communication, tous les tores considérés sont bidimensionnels et de taille 3×3, 3×4, 4×4, 4×5 et 4×6.

La limitation à 4 du nombre de liens de communication d'un processeur rend impossible la connexion directe entre la racine de l'arbre, qui est de degré 4, et le processeur hôte de la VOLVOX. Nous avons donc dû insérer le processeur hôte entre deux processeurs de travail pour pouvoir construire des grilles toriques bidimensionnelles comme indiqué dans la figure 4.

La charge de chaque processeur a été aléatoirement générée selon une loi de distribution uniforme $\mathcal{U}[0, x[$. Donc, les résultats obtenus dépendent de trois paramètres : la borne supérieure x de l'intervalle de distribution, la taille d'une unité de données et la dimension de la grille torique.

Il est évident que le temps des étapes (1) - (8) de l'algorithme (la deuxième phase) ne dépend que de la profondeur de l'arbre. La figure 5 donne le temps moyen de ces étapes pour un échantillon de 12 tests.

Le temps du pas (9), dédié aux échanges de charges, est fortement dépendant de la taille des données à échanger, c'est à dire la taille de l'unité de charge et la taille des paquets de données. La figure 6 donne la durée moyenne de cette étape pour un

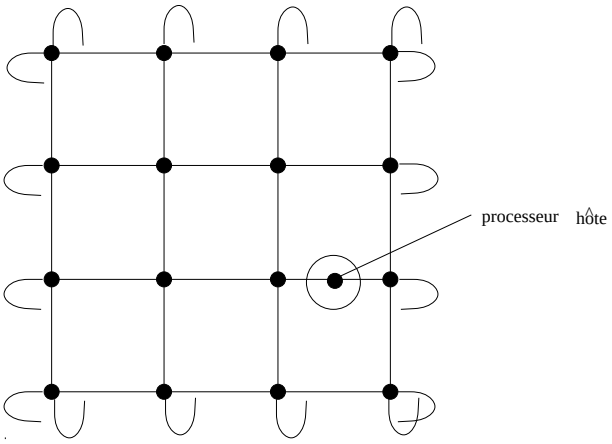

Figure 4. *Configuration de la VOLVOX en grille torique bidimensionnelle de taille* 4×4.

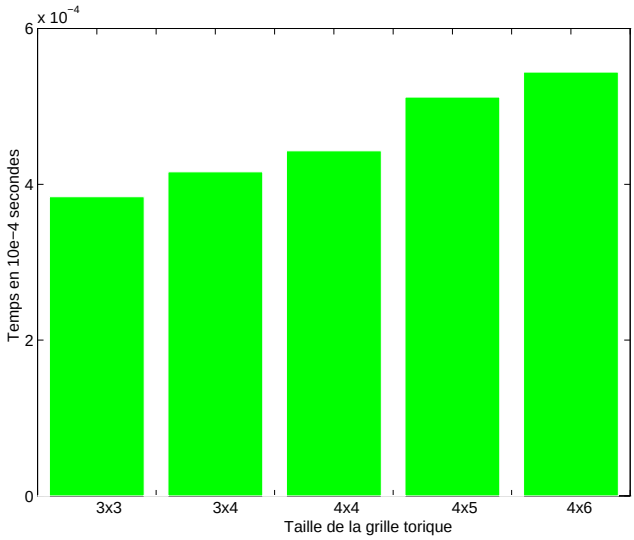

Figure 5. *Le temps nécessaire pour la deuxième phase (il ne dépend que de la topologie).*

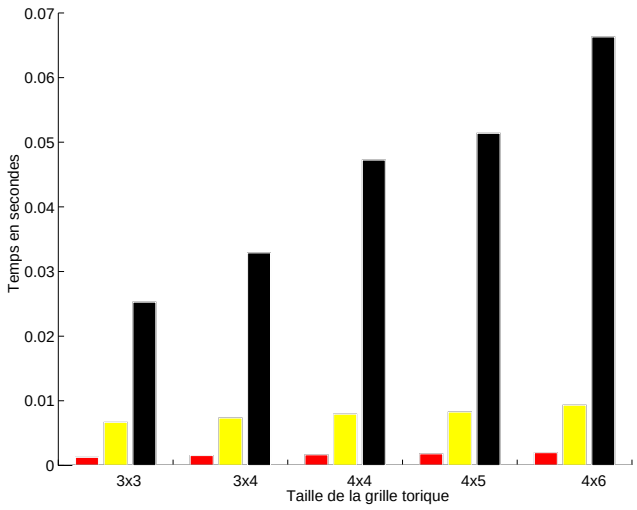

Figure 6. *Variation du temps effectif de répartition en fonction de la taille des unités de charges (1, 10, 100 bytes) pour une distribution de charges* $\mathcal{U}[0, 200[$.

échantillon de 12 tests pour une loi de distribution $\mathcal{U}[0, 200[$ et pour des données de taille unitaire 1, 10 et 100 bytes.

La figure 7 donne le temps de répartition obtenu en variant l'intervalle de génération des charges pour des charges de taille unitaire égale à 32 bytes.

5 Conclusions et travaux futurs

Dans ce papier nous avons présenté une stratégie d'équilibrage de charges de données régulières sur des machines parallèles homogènes MIMD à mémoire distribuée. Cette stratégie a été implantée sur une architecture à base de transputers. Les résultats des tests montrent combien la politique d'échanges de charges, c'est à dire la quantité de données à échanger entre nœuds voisins ainsi que le protocole des échanges, peuvent influencer les performances globales du processus d'équilibrage. Pour la politique d'échanges on peut faire remarquer qu'on peut obtenir de meilleures comme celle donnée par la figure 8.

De telles politiques sont obtenues par améliorations successives à partir d'une politique initiale par réduction du flot de données sur les chemins de poids maximum. L'algorithme présenté dans ce papier peut donc servir pour la génération d'une politique initiale. Aussi un de nos futurs travaux consistera-t-il à l'étude d'une construction efficace d'une politique d'échanges optimale.

Pour le protocole d'échanges proprement dit l'idéal consisterait à pipeliner le flot des données à échanger dans un nœud donné v. C'est à dire que tant que $charge(v) \neq 0$ et $R \neq \emptyset$, v devrait envoyer $charge(v)$ au lieu des quantités exactes $echange(v, u)$, $u \in R$. Ce protocole suppose que tout processeur est capable de réceptions (respectivement d'envois) simultanés sur ses liens, en fait une commutation de données de type wormhole.

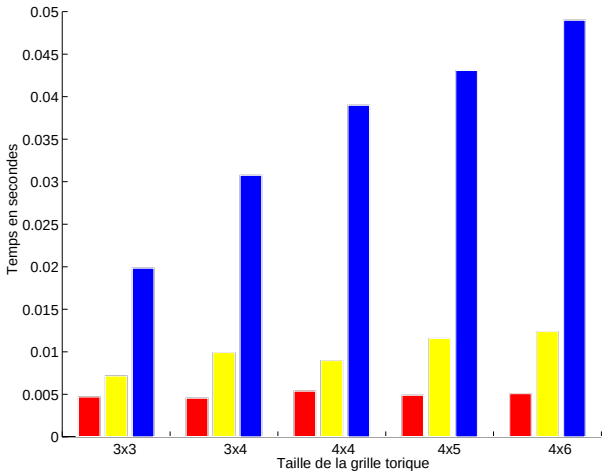

Figure 7. *Variation du temps de répartition en fonction de l'intervalle de distribution ($\mathcal{U}[0,10[, \mathcal{U}[0,100[, \mathcal{U}[0,500[$) des charges de données de taille unitaire égale à 32 bytes.*

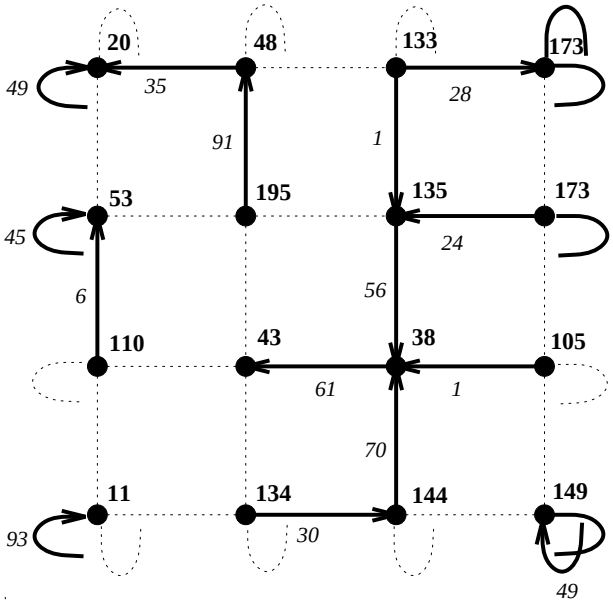

Figure 8. *Un schéma d'équilibrage de coût* ≤ 170.

Nos travaux futurs porteront aussi sur la simulation de ce protocole ainsi qu'une analyse statistique de ses performances.

Références

[BER91] G. Bernard, D. Steve, and M. Simatic. Placement et migration de processus dans les systèmes répartis faiblement couplés. *T.S.I.*, 10(5):355–373, 1991.

[BLE89] Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, November 1989.

[BOI90] J. E. Boillat. Load balancing and Poisson equation in a graph. *Concurrency : Practice and Experience*, 2(4):289–313, December 1990.

[BOI91] J. E. Boillat, F. Brugè, and P.G. Kropf. A dynamic load-balancing algorithm for molecular dynamic simulation on multi-processor systems. *Journal of Computational Physics*, 96:1–14, 1991.

[BRU90] F. Brugè and S. L. Fornili. A distributed dynamic load balancer ans its implementation on multi-transuper systems for molecular dynamics simulation. *Computer Physics Communications*, 60:39–45, 1990.

[CAS88A] Thomas L. Casavant and Jon G. Kuhl. Effects of response and stability on scheduling in distributed computing systems. *IEEE Transactions on Software Engineering*, 14(11):1578–1588, November 1988.

[CAS88B] Thomas L. Cassavant and John G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.

[CYB89] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[GAL83] R. G. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum–weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5:66–75, 1983.

[GER93] D. Gerogiannis and S. C. Orphanoudakis. Load balancing requirements in parallel implementations of image feature extraction tasks. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):994–1013, September 1993.

[HEI94] Alan Heirich. Scalable load balancing by diffusion. Caltech–CS–TR–94–04, California Institut of Technology, 94.

[JAJ92] Joseph Jájá and Kwan Woo Ryu. Load balancing and routing on the hypercube and related networks. *Journal of Parallel and Distributed Computing*, 14:431–435, 1992.

[KOR84] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in a network. *ACM Transactions on Programming Languages and Systems*, pages 380–391, 1984.

[LAV95] Christian Lavault. *Evaluation des algorithmes distribués*. Editions Hermès, 1995.

[MIG92] Serge Miguet. *Redistribution élastique pour équilibrage de charge en imagerie*, chapter 14, pages 229–240. in M. Cosnard and M. Nivat and Y. Robert (ed.) *Algorithmique parallèle*, Masson, Paris, 1992.

[SHI92] Miranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal. Load distributing for locally distributed systems. *IEEE Computer*, 25(12):33–44, December 1992.

[SUB94] R. Subramanian and I. D. Scherson. An analysis of diffusive load–balancing. In *6-th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 220–225, 1994.

[ZAT85] Stephano Zatti. A multivariable information scheme to balance the load in a distributed system. Report No. UCB/CSD 85/234, Computer Science Division, University of California Berkeley, May 1985.